Privacy-Preserving Graph Encryption for Approximate Constrained Shortest Distance Queries

Meng Shen*†, Siqi Chen*, Liehuang Zhu*, Renyi Xiao‡, Ke Xu§, and Xiaojiang Du¶
* School of Computer Science, Beijing Institute of Technology, Beijing, China
† State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, 100878, China
‡ National Natural Science Foundation of China, Beijing, China
§ Department of Computer Science, Tsinghua University & BNRist, Beijing, China
¶Department of Computer and Information Sciences, Temple University, Philadelphia, USA

***Abstract*—Constrained shortest distance (CSD) queries are a valuable extension of the traditional pairwise shortest distance computation over graph-structured data, where the answers to the queries should fulfill a *cost* constraint (e.g., the toll payment in road networks). With the popularity of cloud computing, data owners have a strong desire to migrate their privacy-sensitive graphs to remote servers without losing the ability to query them. Existing graph encryption schemes cannot provide security guarantees for CSD queries.**

In this paper, we present `Acro`, a graph encryption scheme, which executes approximate CSD queries securely. The homomorphic encryption and the symmetric-key primitives are applied to our scheme. Through a security analysis, we prove that `Acro` meets the security definition of CQA2-security. The prototype of `Acro` is implemented and evaluated using real datasets. The results show that our proposal outperforms a state-of-the-art baseline in terms of query accuracy at the cost of enlarging query completion time.

***Index Terms*—Graph encryption, constrained shortest distance querying, privacy-preserving, searchable encryption**

I. Introduction

Graph-structured data is widely used in numerous domains, such as road networks, online social networks, web graphs and so on. A number of systems, e.g., Pregel and GraphLab, have been proposed to facilitate graph management and operations. Due to the economic and efficient nature of cloud computing, lots of users tend to migrate their databases to the cloud, including the graph data owners. Furthermore, users also show a natural worry about the security and privacy in the cloud.

Encrypting graphs is an effective way for privacy protection, but it also results in the loss of ability to be queried. To address this problem, structured encryption [3, 13] was proposed to allow for some forms of computation and query over encrypted data. Graph encryption is a special situation in the structured encryption, which provides data owners with the ability to conduct queries over cloud-based encrypted data.

Unlike previous studies on graph encryptions for shortest distance queries [3, 10], we aim at designing an efficient graph encryption scheme that supports *constraint* shortest distance (CSD) queries. The goal of a CSD query is to achieve the shortest *distance* from a given source to a destination (s,t) with *cost* c under the cost constraint θ. Many applications use CSD queries as a basic operation [11]. For instance, it can find the minimum distance (e.g., miles) between two cities in road networks within a total budget of toll payment.

CSD querying on unencrypted graphs is an extension of the traditional shortest distance querying [7, 16]. Since the CSD querying is proven to be a NP-hard problem [7], Lots of existing studies (e.g., [7, 12, 16]) propose approximate solutions, such as the α-approximation where the resulting distance is guaranteed to be shorter than α times of the optimal one. There are few studies on privacy-preserving CSD querying over encrypted graphs. `Connor` [12] solves α-approximate CSD queries while considering privacy protection. However, it relies on order-revealing encryption (ORE), which reveals order information of costs and distances and also results in querying answers that violate the cost constraint.

In this paper, we present `Acro`, a graph encryption scheme, which executes approximate CSD queries over cloud-based encrypted graphs. Similar to existing studies, our scheme employs the 2-hop cover labeling index (2HCLI) [10, 16], a method that pre-computing the distance oracle to simplify calculations of the distance between vertices. To achieve computational security, a specific pseudo-random function (PRF) is used to encrypt the vertices of the graph. Somewhat homomorphic encryption (SWHE) [1] is also applied to the encryption of distances and costs.

Our contributions can be summarized as follows:

1) We present a new graph encryption scheme named `Acro` for approximate CSD querying over cloud-based encrypted graphs. The cryptographic primitives used in our construction ensure that `Acro` obtains exactly the same results as plain graph indexing.
2) We propose a split-server model where two non-collusive servers are employed to implement privacy-preserving queries together: one executes distance computation and the other executes cost filtering.
3) We present security analysis on `Acro` to prove that it conforms to the security definition of CQA2-security [3]. We evaluate `Acro` using real datasets to demonstrate its effectiveness and accuracy.

The content of this paper can be generalized as follows: We briefly describe the related work in Section II, then introduce CSD query backgrounds in Section III. The problem of secure approximate CSD query is formally defined in Section IV, followed by the description of `Acro` system model in Section V. The security analysis and experimental results of `Acro` are in Section VI and Section VII respectively. We give our conclusion in Section VIII.

II. Related Work

During the past decade, there are lots of works about constrained shortest distance queries over plaintext graph. Tsaggouris and Zaroliag [15] followed the general idea of Dijkstra's algorithm and proposed an index-free method for approximating constraint shortest paths. Due to the inefficiency of Dijkstra's algorithm on large-scale graphs, researchers resort to index-based schemes that use distance oracles to answer shortest distance queries. Wang et al. [16] proposed an efficient overlay indexing structure to answer approximate constrained shortest path queries on large-scale road networks.

With the rise of privacy concerns in the cloud computing era, many research efforts have been made to develop schemes that allow querying graphs privately [8, 9]. The notion of graph encryption was provided by Chase and Kamara [3]. Following that, Meng et al. [10] presented three models based on distance oracles to answer approximate shortest distance querying, and their models were proven secure against a curios-but-honest server. Shen et al. [12] considered the constraint shortest distance querying over encrypted graphs, and proposed a scheme named `Connor` based on ORE and SWHE primitives.

Our work is closely related with the study in [12], but with significant differences. We propose a split-server model for α-CSD queries, which decouples the cost filtering functionality in the server $\mathcal{P}$ from the index storage server $\mathcal{C}$. As the ORE has been demonstrated to be vulnerable to leakage-abuse attacks [2], our scheme can achieve better security by using SWHE primitive, i.e., without revealing order relationship in the vertex label sets. Unlike the scheme in [12] which results in answers violating cost constraint, the SWHE primitive in our solution can guarantee the correctness of query answers.

III. Background

A. Approximate CSD GenQuery

We define V as a vertex set and E as an edge set, For a given directed graph $G = (V, E)$, each edge $e \in E$ has two attributes: a *cost* $c(e) \geq 0$ and a *distance* $d(e) \geq 0$. Here, the cost is regarded as the constraint.

The *path* is defined as the set of edges between a pair of vertices, denoted by P. The distance and cost of a path are the total distance and cost of edges composing the path. For a path $P = (e_1, e_2, \ldots, e_k)(k = |P|)$, its distance $dist(P)$ and cost $cost(P)$ can be expressed as $dist(P) = \sum_{i=1}^{k} d(e_i)$ and $cost(P) = \sum_{i=1}^{k} c(e_i)$. Table I summarizes the notations used in this paper.

Definition 1. *(α-CSD QUERY [12]).* For a given graph $G = (V, E)$, an α-CSD query $q = (s, t, \theta, \alpha)$ with a source-destination pair (s, t), a cost constraint θ, and an approximation ratio α, α-CSD query finds a path P so that $cost(P) \leq \theta$ and $dist(P) \leq \alpha \cdot dist(P^*)$. Note that $dist(P^*)$ is the optimal solution of the CSD query with the same s, t, and θ.

TABLE I: List of Notations

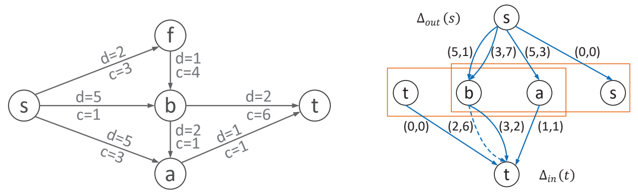
Notation	Meaning
$G = (V, E)$	A graph with a vertex set and an edge set
$d(e), c(e)$	Distance and cost of an edge $e \in E$
$d_{u,v}, c_{u,v}$	Distance and cost from vertex u to vertex v
$q = (s, t, \theta, \alpha)$	Source, destination, cost constraint and approximation ratio in an α-CSD query q
$\Delta, \widetilde{\Delta}$	Plain and encrypted graph index
$Set_{in}(v)$	In-label sets for vertex v
$Set_{out}(v)$	Out-label sets for vertex v
$\mathcal{C}$, $\mathcal{P}$	The cloud server and the cost filtering server
Q	The set of queries
$\tau_{s,t}$	The query token
λ	Security parameter
z	Input length of symmetric encryption
$\tilde{\theta}$	Encrypted cost constraint perceived by $\mathcal{P}$
$\tilde{A}$	Encrypted distance vector returned to *user*

B. Graph Index

Our graph encryption scheme is constructed on a well-known distance oracle, 2-hop cover labeling index (2HCLI)[4, 5, 12, 16]. For a given directed graph $G = (V, E)$, each vertex $u \in V$ involves two sketches, an in-label set $Set_{in}(u)$ and an out-label set $Set_{out}(u)$. Take $Set_{in}(u)$ as an example. To enable constrained shortest distance queries, each entry in $Set_{in}(u)$ is represented by a 3-tuple $(w, d_{w,u}, c_{w,u})$, where $d_{w,u}$ and $c_{w,u}$ indicate the shortest distance and the corresponding cost from a vertex $w \in V$ to u, respectively. Note that the existence of different criteria (i.e., distance and cost) may result in multiple entries with different distance and cost values for the same vertex w in $Set_{in}(u)$.

To reduce the size of 2HCLI, it is important to filter out redundant entries in out-label sets and in-label sets. To facilitate this process, we can use the definition of *α-dominance*. We refer readers to the literature [16] for more details.

Definition 2. *(α-Dominance).* Given two paths P_1 and P_2 with the same source and destination, P_1 α-dominates P_2 iff $c(P_1) \leq c(P_2)$ and $d(P_1) \leq \alpha \cdot d(P_2)$.



(a) A simple graph (b) 2HCLIs

Fig. 1: Examples illustrating CSD queries and 2HCLIs.

Example 1. *Fig. 1(a) shows an example graph consisting of five vertices. For each edge, its distance and cost values are marked alongside it. Given an α-CSD query $q = (s, t, 4, 1.5)$, the corresponding plain graph index is depicted in Fig. 1(b).*

*According to Definition 2, the path $P_{bt}^1 = \langle b, a, t\rangle$, whose $(dis, cost)$ pair is (3,2), and it α-*dominates *the other path $P_{bt}^2 = \langle b, t\rangle$, which has (2,6) of the $(dis, cost)$ pair. Thus, we can remove the entry corresponding to P_{bt}^2 from $Set_{in}(t)$, as represented by a dashed arrow in Fig. 1(b). However, neither of the two paths from vertex s to vertex b (e.g., the two solid arrows) can be removed from $Set_{out}(s)$. The answer of the α-CSD query $q = (s, t, 4, 1.5)$ is $d_{s,a}+d_{a,t} = 6$ with $c_{s,a}+c_{a,t} = 4$, which is exactly the answer of the CSD query.*

IV. Problem Description

We give the definition of a privacy-preserving α-CSD query problem in this section, including descriptions of the system model and the security model.

A. System Model

In order to enable secure α-CSD queries, the construction of our system architecture mainly involves three entities: the user $\mathcal{U}$, the cloud server $\mathcal{C}$, and the cost filtering server $\mathcal{P}$. Fig. 2 shows a complete construction of our system model.

In the initialization phase, the *user* constructs a secure searchable graph index and delivers the encrypted index with the encrypted graph together to the sever $\mathcal{C}$. Meanwhile, the *user* delivers the key pair $(pkey, skey)$ to the sever $\mathcal{P}$.

During the query stage, after receiving the query token from the *user*, the sever $\mathcal{C}$ selects a set W of intermediate vertices from the encrypted index that connects the source s and the destination t. The server $\mathcal{P}$ performs the cost filtering to filter out cost pairs $(c_{s,w}, c_{w,t})$ for $w \in W$ where $c_{s,w}+c_{w,t}$ exceeds the cost constraint. Next, the sever $\mathcal{C}$ replies to the *user* with the result after filtering, which includes the candidate distances satisfying the cost constraint. Finally, the *user* selects the minimum distance value as the answer of the query.

The following definition describes our graph encryption scheme.

Definition 3. *(GRAPH ENCRYPTION).* Our scheme consists of four algorithms: $GenKey$, $GenIndex$, $GenQuery$ and $Filtering$:

- $GenKey(\lambda) \rightarrow (K, pkey, skey)$**.** This algorithm generates the secret key. It requires a security parameter λ to achieve a secret key K and a key pair, denoted as $(pkey, skey)$ representing the public and private keys respectively.
- $GenIndex(\alpha, K, pkey, skey, G) \rightarrow \widetilde{\Delta}$**.** $GenIndex$ is a graph encryption algorithm to achieve a secure index $\widetilde{\Delta}$. Note that α is an approximation ratio.
- $GenQuery((K, pkey, skey, q), \widetilde{\Delta}) \rightarrow (dist_q, \perp)$**.** The algorithm requires a user who masters $(K, pkey, skey, q)$ and a server $\mathcal{C}$ that masters $\widetilde{\Delta}$ to achieve the distance $dist_q$ as the result for users, and an end symbol $\perp$ for the server $\mathcal{C}$.

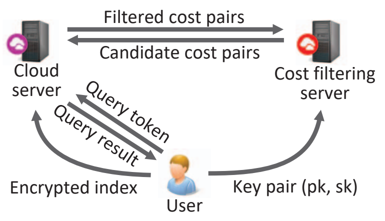


Fig. 2: The architecture of privacy-preserving CSD querying.

- $Filtering(pkey, skey, L_c, \tilde{\theta}) \rightarrow \tilde{M}$. The filtering algorithm requires a key pair $(pkey, skey)$, a vector L_c. and an encrypted cost constraint $\tilde{\theta}$. The purpose is to determine whether the elements in the vector satisfy the cost constraints.

B. Security Model

We use the same security definition as symmetric searchable encryption (SSE) in our scheme, since the graph encryption is included in SSE [6, 14]. Our security definition is also similar to the security definition proposed in [3, 6], which is related to chosen-query attack security (CQA2-security).

We assume that the server $\mathcal{C}$ and $\mathcal{P}$ is *honest-but-curious* and non-colluding adversaries, which would honestly follow the predefined algorithms, but may try to learn extra sensitive information from their views of the scheme.

To observe the entire scheme, leakages may happen in three phases: $GenIndex$, $GenQuery$ and $Filtering$. The server $\mathcal{C}$ handles $GenIndex$ and $GenQuery$ algorithm, which is able to access some sensitive information such as the size of $\widetilde{\Delta}$, the total number of sketches and the sequence of queries. The server $\mathcal{P}$ is responsible for handling candidate vectors and cost constraints, which is possible to reveal the relationship between each cost and constraints.

There has been a complete discussion in [12] about the security under the server $\mathcal{C}$ adversary, so we focus on the security analysis under the server $\mathcal{P}$ adversary. We consider a semi-honesty adversary $\mathcal{P}$, probably leaking the information at the cost filtering phase. The leakage in the cost filtering process can be defined as a leakage function $\mathcal{L}_{Filter}$.

V. Construction of Acro

In this section, we describe the construction of our scheme Acro for privacy-preserving α-CSD queries.

A. Privacy-preserving α-CSD Querying

The graph encryption scheme for α-CSD querying consists of 4 components, namely $GenKey$, $GenIndex$, $GenQuery$ and $Filtering$. The key generation procedure $GenKey$ is as follows: For a given security parameter λ, the *user* generates a secret key K and a pair of key $(pkey, skey)$ in random for the SWHE.

$$pr_1 : \{0,1\}^{\lambda} \times \{0,1\}^{*} \rightarrow \{0,1\}^{\lambda} \qquad (1a)$$

$$pr_2 : \{0,1\}^{\lambda} \times \{0,1\}^{*} \rightarrow \{0,1\}^{\lambda+2z} \qquad (1b)$$

We introduce two functions pr_1 and pr_2, both of which are pseudo-random functions, used as defined in the following equation, where λ is the security parameter. The SWHE scheme adopted in our construction is the SWHE instantiation in [1] with an output length of z.

The $GenIndex$ process is depicted in Algo. 1. For a given graph $G = (V, E)$ and an approximation ratio α, the *user* firstly generates the 2HCLI Δ of the graph G, then encrypts the sketches of each vertex $v \in V$, and finally gets the encrypted graph index $\widetilde{\Delta}$.

Note that if we simply store all entries in Set_{out} or Set_{in} in the same position of the dictionary, it would reveal *static* information of the encrypted index when no query is issued, such as the size of each encrypted sketch, and the number of vertices between $Dict_{in}(u)$ and $Dict_{out}(v)$, implying the quantity of vertices connecting u to v.

Algorithm 1 GenIndex Algorithm for Graph Encryption

```
Input: $(K, pkey, skey, \alpha, G)$
Output: $\widetilde{\Delta}$.
 1: Produce the 2HCLI $\Delta = \{Set_{in}, Set_{out}\}$ of the graph $G$.
 2: Set two dictionaries $Dict_{in}$ and $Dict_{out}$.
 3: for each $u \in G$ do
 4:     Set $S_{out,u} = pr_1(K, u||1)$, $T_{out,u} = pr_1(K, u||2)$, $S_{in,u} = pr_1(K, u||3)$, and $T_{in,u} = pr_1(K, u||4)$.
 5:     Set a counter $\varepsilon = 0$
 6:     for each $(v, d_{u,v}, c_{u,v}) \in Set_{out}(u)$ do
 7:         Calculate $V = pr_1(K, v||0)$.
 8:         Calculate $D_{u,v} = \text{SWHE.Enc}(pkey, d_{u,v})$.
 9:         Calculate $C_{u,v} = \text{SWHE.Enc}(pkey, c_{u,v})$.
10:         Set $T_{out,u,v} = pr_1(T_{out,u}, \varepsilon)$ and $S_{out,u,v} = pr_2(S_{out,u}, \varepsilon)$.
11:         Calculate $\Psi_{u,v} = S_{out,u,v} \oplus (V||D_{u,v}||C_{u,v})$.
12:         Set $Dict_{out}[T_{out,u,v}] = \Psi_{u,v}$.
13:         Set $\varepsilon = \varepsilon + 1$.
14:     end for
15:     Execute the same process repeatedly for the sketch in $Set_{in}(u)$ and get $Dict_{in}[T_{in,u,v}]$,
16:     Set $T_{in,u,v} = pr_1(T_{in,u}, \varepsilon)$ and $S_{in,u,v} = pr_2(S_{in,u}, \varepsilon)$
17:     Calculate $\Psi_{u,v} = S_{in,u,v} \oplus (V||D_{u,v}||C_{u,v})$.
18: end for
19: return $\widetilde{\Delta} = \{Dict_{in}, Dict_{out}\}$.
```

To prevent such information leakage, each encrypted sketch is decomposed into individual entries and stored in the dictionary in sequence. For each entry in $Set_{out}(u)$, a counter ε is utilized to generate a unique pair $T_{out,u,v}$ and $S_{out,u,v}$ (line 10). $T_{out,u,v}$ is used as a position indicator in $Dict_{out}$ (line 12), while $S_{out,u,v}$ is used to protect the content of each encrypted entry $(V||D_{u,v}||C_{u,v})$. More precisely, an XOR operation between $(V||D_{u,v}||C_{u,v})$ and $S_{out,u,v}$ can make the resulting $\Psi_{u,v}$ indistinguishable, even when two entries of $(V||D_{u,v}||C_{u,v})$ are associated with the same vertex V (line 11). Similar operations can be taken for each sketch in $Set_{in}(u)$ to obtain $Dict_{in}$ (line 15). This way, we can ensure that from the static encrypted graph index $\widetilde{\Delta}$, an adversary can learn none of the following sensitive information: the sketch of each individual vertex, the common vertices between a pair of vertex sketches, and the distance or cost values in an entry.

Example 2. *Fig. 3 shows an example of the dictionary structure for the out-label set $Set_{out}(s)$ in Fig. 1(b). Originally, $Set_{out}(a)$ has 4 entries. In the dictionary $Dict_{out}(a)$, each entry of $Set_{out}(s)$ is stored separately with a size of one, ordered by the counter ε.*

The $GenQuery$ process is shown in Algo. 2. Assume that a *user* requests an α-CSD query with (s, t, θ). The *user* generates the corresponding query token $\tau_{s,t}$ and delivers it to the server $\mathcal{C}$ (lines 1-2). Based on the parameters in $\tau_{s,t}$, server $\mathcal{C}$ obtains all entries associated with vertices s and t, which are stored in L_s and L_t, respectively (line 6-21). This process is conducted as a reverse operation of constructing the encrypted graph index.

Once L_s and L_t are obtained, $\mathcal{C}$ chooses the vertices that appears in both L_s and L_t, merges their distances and costs, and adds the processed pairs into L (lines 25-27). Then, $\mathcal{C}$ selects a random number r and encrypts individual costs and the cost constraint θ with the SWHE, which protects their real values from server $\mathcal{P}$. Server $\mathcal{C}$ calculates the sum of r and c, as well as r and θ, whose correctness follows homomorphic properties of the SWHE. L_c and $\tilde{\theta}$ are then sent to server $\mathcal{P}$.

$\mathcal{P}$ performs cost filtering as depicted in Algorithm 3 and returns a vector $\tilde{M}$. For each element in $\tilde{M}$ and each distance, server $\mathcal{C}$ calculates SWHE.Eval$(\times, \tilde{M}_i, D_i)$ and sends the result vector $\tilde{A}$ to the *user*. The *user* decrypts $\tilde{A}_i \in \tilde{A}$ with $skey$ and selects the minimum *non-zero* value as the final answer.

The $Filtering$ process is illustrated in Algorithm 3. Server $\mathcal{P}$ decrypts L_c and $\tilde{\theta}$ received from the server $\mathcal{C}$ with $skey$. Although $\mathcal{P}$ is unaware of the exact value of c, it can determine the relationship between c and the cost constraint θ: If $c + r$ is less than $\theta + r$, the cost c is eligible; otherwise the cost c violates the constraint. Then, it encrypts the vector M with the SWHE and returns the vector $\tilde{M}$ to server $\mathcal{C}$.

Example 3. *Fig.4 shows a schematic of the* Filtering *process. It assigns each item in L_c with a number (1 for valid and 0 for invalid) and records the results in the encrypted vector $\tilde{M}$.*

VI. Security Analysis

This section presents the security analysis of our graph encryption scheme Acro. We discuss the leakage observed by the server $\mathcal{C}$ and prove that Acro is secure under the CQA2-security model.

As described in Section IV, we discuss the leakage on the server $\mathcal{P}$ in the cost filtering process. The leakage function $\mathcal{L}_{Filter}$ represents the cost pattern leakage during the cost filtering phase. The cost pattern leakage reveals the relationship

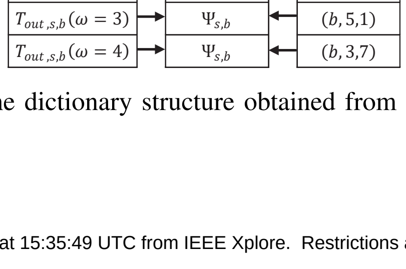


Fig. 3: The dictionary structure obtained from *GenIndex*

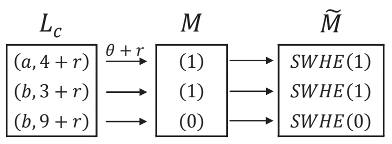


Fig. 4: Filtration algorithm full schematic (θ=4).

Algorithm 2 GenQuery Algorithm for Graph Encryption

```
Input: The user: $(K, pkey, skey)$, the query $q = (s, t, \theta)$, The server $\mathcal{C}$: $\widetilde{\Delta}$.
Output: The user: $dist_q$    The server $\mathcal{C}$: $\perp$.
 1: The user generates $S_{out,s} = pr_1(K, s||1)$, $T_{out,s} = pr_1(K, s||2)$,
    $S_{in,t} = pr_1(K, t||3)$ , $T_{in,t} = pr_1(K, t||4)$ and $T_\theta$ =
    SWHE.Enc($pkey, \theta$)
 2: The user sends $\tau_{s,t} = (S_{out,s}, T_{out,s}, S_{in,t}, T_{in,t}, T_\theta)$ to the server
    $\mathcal{C}$.
 3: $\mathcal{C}$ parses $\tau_{s,t}$ as $(S_{out,s}, T_{out,s}, S_{in,t}, T_{in,t}, T_\theta)$.
 4: $\mathcal{C}$ generates a set $L_s$, a counter $\varepsilon = 0$.
 5: $\mathcal{C}$ calculates $T_{out,s,v} = pr_1(T_{out,s}, \varepsilon)$.
 6: while $Dict_{out}[T_{out,s,v}] \neq \perp$ do
 7:     $\mathcal{C}$ calculates $S_{out,s,v} = pr_2(S_{out,s}, \varepsilon)$.
 8:     $\mathcal{C}$ performs $(V||D_{s,v}||C_{s,v}) = \Psi_{s,v} \oplus S_{out,s,v}$.
 9:     $\mathcal{C}$ adds $(V, D_{s,v}, C_{s,v})$ into $L_s$.
10:     Set $\varepsilon = \varepsilon + 1$.
11:     $\mathcal{C}$ calculates $T_{out,s,v} = pr_1(T_{out,s}, \varepsilon)$.
12: end while
13: $\mathcal{C}$ generates a set $L_t$, a counter $\varepsilon = 0$.
14: $\mathcal{C}$ calculates $T_{in,v,t} = pr_1(T_{in,t}, \varepsilon)$.
15: while $Dict_{in}[T_{in,v,t}] \neq \perp$ do
16:     $\mathcal{C}$ calculates $S_{in,v,t} = pr_2(S_{in,t}, \varepsilon)$.
17:     $\mathcal{C}$ performs $(V||D_{v,t}||C_{v,t}) = \Psi_{v,t} \oplus S_{in,v,t}$.
18:     $\mathcal{C}$ adds $(V, D_{v,t}, C_{v,t})$ into $L_t$.
19:     Set $\varepsilon = \varepsilon + 1$.
20:     $\mathcal{C}$ calculates $T_{in,v,t} = pr_1(T_{in,t}, \varepsilon)$.
21: end while
22: $\mathcal{C}$ generates a vector $L$ and a vector $L_c$.
23: for every encrypted identifier $v$ that is in both $L_s$ and $L_t$ do
24:     pick the entries $(V, D_{s,v}, C_{s,v}), (V, D_{v,t}, C_{v,t})$ from $L_s$ and $L_t$.
25:     $\mathcal{C}$ calculates $D$=SWHE.Eval$(+, D_{s,v}, D_{v,t})$.
26:     $\mathcal{C}$ calculates $C$=SWHE.Eval$(+, C_{s,v}, C_{v,t})$.
27:     $\mathcal{C}$ adds $(V, D, C)$ into $L$.
28:     $\mathcal{C}$ selects a random number $r$.
29:     $\mathcal{C}$ calculates $R$=SWHE.Enc$(pkey, r)$.
30:     $\mathcal{C}$ calculates $\widetilde{C}$=SWHE.Eval$(+, C, R)$.
31:     $\mathcal{C}$ calculates $\widetilde{\theta}$=SWHE.Eval$(+, T_\theta, R)$.
32:     $\mathcal{C}$ adds $(V, \widetilde{C})$ into $L_c$.
33: end for
34: $\mathcal{C}$ delivers the $\widetilde{\theta}$ and the $L_c$ to the server $\mathcal{P}$, who executes cost filtering
    in Algorithm 3 and returns a vector $\widetilde{M}$.
35: for each element $\widetilde{M}_i \in \widetilde{M}$ and each entry $D_i \in L$ do
36:     $\mathcal{C}$ calculates $\widetilde{A}_i$=SWHE.Eval$(\times, \widetilde{M}_i, D_i)$.
37: end for
38: $\mathcal{C}$ sends $\widetilde{A}$ to the user.
39: Decrypting $\widetilde{A}$ with $skey$ to get the minimum distance as the answer.
```

between each cost and the constraint, as well as the size of the candidate vector L_c. Note that $\mathcal{L}_{Filter}$ does not reveals the real value of costs and the cost constraint, since L_c and $\tilde{\theta}$ are perturbed with a random noise by serber $\mathcal{C}$.

Given an encrypted cost vector L_c and a masked cost constraint $\tilde{\theta}$, the cost pattern leakage function $\mathcal{L}_{Filter}(L_c, \tilde{\theta})$ is defined as $(\boldsymbol{n}, l_{\boldsymbol{n}})$, where $\boldsymbol{n}$ is a vector with a length $l_{\boldsymbol{n}} = |L_c|$ and each element is 1 or 0, representing that the cost constraint is satisfied or violated.

Our idea is to construct a simulator $\mathcal{S}$. For a given $\mathcal{L}_{Filter}$, $\mathcal{S}$ constructs a fake candidate vector L_c^* and a cost constraint $\tilde{\theta}^*$. If the adversary cannot distinguish between the real one and the fake one, our cost filtering process is secure enough against the attack.

Algorithm 3 Filtering Algorithm for Graph Encryption

```
Input: The Sever $\mathcal{P}$: $(pkey, skey, L_c, \tilde{\theta})$.
Output: The vector $\widetilde{M}$
 1: $\mathcal{P}$ initializes an empty vector $M$ with the same length of $L_c$.
 2: $\mathcal{P}$ decrypts $L_c$ and $\tilde{\theta}$ with $skey$ to get $c + r$ and $\theta + r$.
 3: for each entry in the $L_c$ do
 4:     if the $c + r$ is less than $\theta + r$ then
 5:         $\mathcal{P}$ assigns 1 to $M_i$
 6:     else
 7:         $\mathcal{P}$ assigns 0 to $M_i$.
 8:     end if
 9: end for
10: $\mathcal{P}$ calculates $\widetilde{M}$=SWHE.Enc$(pkey, M)$.
11: $\mathcal{P}$ sends $\widetilde{M}$ to the sever $\mathcal{C}$.
```

TABLE II: Datasets of the experiments

Dataset	Nodes	Edges	Storage
D1: Email-EuAll	21,721	34,351	335KB
D2: soc-Epinions1	6,506	47,062	418KB
D3: p2p-Gnutella04	10,876	39,994	422KB
D4: p2p-Gnutella25	22,687	54,705	632KB

For any L_c^* and $\tilde{\theta}^*$, the server $\mathcal{P}$ only executes comparisons between $(c + r)^*$ and $(\theta + r)^*$, which is not able to get more valid information. So if the cryptography primitives pr_2, pr_1 and the SWHE are secure, the fake vector L_c^* and cost constraint $\tilde{\theta}^*$ cannot br distinguished from the real ones. Due to the complete security of cryptography primitives such as pr_2, pr_1 and the SWHE, we have enough security at the cost filtering phase.

VII. Experimental Evaluation

In this section, we conduct experiments using real datasets to evaluate the performance of the proposed graph encryption scheme.

A. *Experimental GenIndex*

Testbed. A prototype of Acro is implemented in C++, where the 2HCLI and the SWHE algorithm in our construction follow the implementation introduced in [16] and [1], respectively. All the basic cryptographic primitives use the OpenSSL library and the security parameter λ is set to be 128. We conduct the experiments on machines with different configurations: The *user* runs on a laptop with Intel Core CPU at 2.70 GHz and 4GB RAM, while the two servers $\mathcal{C}$ and $\mathcal{P}$ run on separate machines with Intel Xeon processor at 2.6GHz and 8GB RAM.

Datasets. As shown in Table II, 4 directed real-world graphs are selected for evaluation, which are publicly available from the Stanford SNAP website[1]. The cost and distance values of each edge are randomly generated following a uniform distribution between 1 and 100.

For each graph in Table II, 200 queries are generated with randomly selected origins and destinations. For each query,

[1]http://snap.stanford.edu/data/

TABLE III: The query time over different datasets

Dataset / Time (ms)	Connor	Acro
D1: Email-EuAll	16.15	155.29
D2: soc-Epinions1	32.63	238.62
D3: p2p-Gnutella04	24.99	1127.13
D4: p2p-Gnutella25	65.21	1512.72

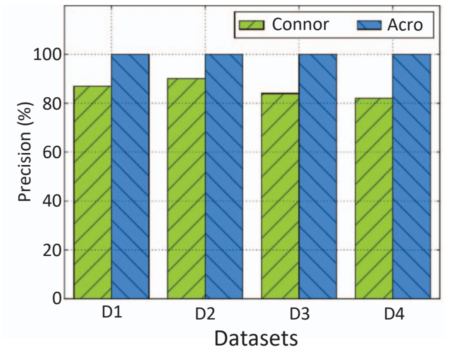


Fig. 5: The query precision over different datasets.

50 values of θ are randomly generated with an additional guarantee that valid answers always exist given a specific θ. The approximation ratio α is regarded as a system parameter and is set to be 1.5.

B. *Evaluation of Accuracy and Efficiency*

Accuracy. We compare `Acro` with `Connor`, as the latter is the most closely related state-of-the-art approach. Compared with `Connor`, the query answer obtained from `Acro` for each query is exactly the same as the one derived from plain graph index. To demonstrate the difference, we define an attribute named *precision*. Given a set of queries Q, the precision of a specific encryption scheme (i.e., `Acro` or `Connor`) is defined as the percentage of the queries whose answers are exactly the same with the ones obtained from the plain graph index. Fig. 5 depicts the precision of the two schemes over different datasets. The precision of our method remains 100% over different datasets, while the one of `Connor` is around 80%. The main reason lies in that the ORE in `Connor` cannot accurately identify the relationship between each cost pair (e.g., $(c_{s,w}, c_{w,t})$) and the cost constraint θ and thereby may return answers that violate the cost constraint.

Efficiency. The query time means the duration from the query is issued until the result is received. The average query time over different datasets is exhibited in Table III. For each dataset, the query time of our scheme is higher than that of `Connor`, since we utilize the SWHE primitive rather than the ORE primitive in the cost filtering process, which achieves higher accuracy and privacy guarantees at the cost of increasing computational complexity. The query time of both methods increase with the increasing size of the graphs.

VIII. Conclusion

In this paper, we propose `Acro`, a practical graph encryption scheme which can achieve approximating CSD queries over encrypted graphs. We designed a split-server model, where one executes distance computation and the other performs cost filtering. Security analysis showed that `Acro` is secure under the CQA2-security. The experimental evaluation using real-world graphs demonstrated the effectiveness and accuracy of our scheme. In future work, we will further improve query efficiency.

Acknowledgements

This work is partially supported by the National Key Research and Development Program of China under Grant 2018YFB0803405, the National Natural Science Foundation of China under Grants 61602039 and 61872041, the Beijing Natural Science Foundation under Grant 4192050, the China National Funds for Distinguished Young Scientists under Grant No. 61825204, the Beijing Outstanding Young Scientist Project, and CCF-Tencent Open Fund WeBank Special Funding.

References

[1] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, pages 325–341. Springer, 2005.
[2] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 668–679, New York, NY, USA, 2015. ACM.
[3] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, pages 577–594. Springer, 2010.
[4] J. Cheng and J. X. Yu. On-line exact shortest distance query processing. In *EDBT*, pages 481–492, 2009.
[5] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *Siam Journal on Computing*, 32(5):937–946, 2002.
[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proc. of ACM CCS*, pages 79–88, New York, NY, USA, 2006. ACM.
[7] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations research*, 17(1):36–42, 1992.
[8] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography*, pages 457–476. Springer, 2013.
[9] Y. Li, H. Shen, C. Lang, and H. Dong. Practical anonymity models on protecting private weighted graphs. *Neurocomputing*, 218(C):359–370, 2016.
[10] X. Meng, S. Kamara, K. Nissim, and G. Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *ACM CCS*, pages 504–517, New York, NY, USA, 2015. ACM.
[11] M. Shen, B. Ma, L. Zhu, X. Du, and K. Xu. Secure phrase search for intelligent processing of encrypted data in cloud-based IoT. *IEEE Internet of Things Journal*, 6(2):1998–2008, 2018.
[12] M. Shen, B. Ma, L. Zhu, R. Mijumbi, X. Du, and J. Hu. Cloud-based approximate constrained shortest distance queries over encrypted graphs with privacy protection. *IEEE Transactions on Information Forensics and Security*, 13(4):940–953, April 2018.
[13] M. Shen, X. Tang, L. Zhu, X. Du, and M. Guizani. Privacy-preserving support vector machine training over blockchain-based encrypted IoT data in smart cities. *IEEE Internet of Things Journal*, 2019.
[14] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *NDSS*, 2014.
[15] G. Tsaggouris and C. Zaroliagis. *Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-linear Objectives with Applications*. Springer Berlin Heidelberg, 2006.
[16] S. Wang, X. Xiao, Y. Yang, and W. Lin. Effective indexing for approximate constrained shortest path queries on large road networks. *PVLDB*, 10(2):61–72, 2016.